iOS Programmierung

(mit Swift)

Peter Braun, Florian Bachmann & Andreas Wittmann
@pe_braun @florianbachmann @anwittmann

Deutsche Telekom AG
FHWS - Hochschule für angewandte Wissenschaften Würzburg-Schweinfurt

#FHWSSwift

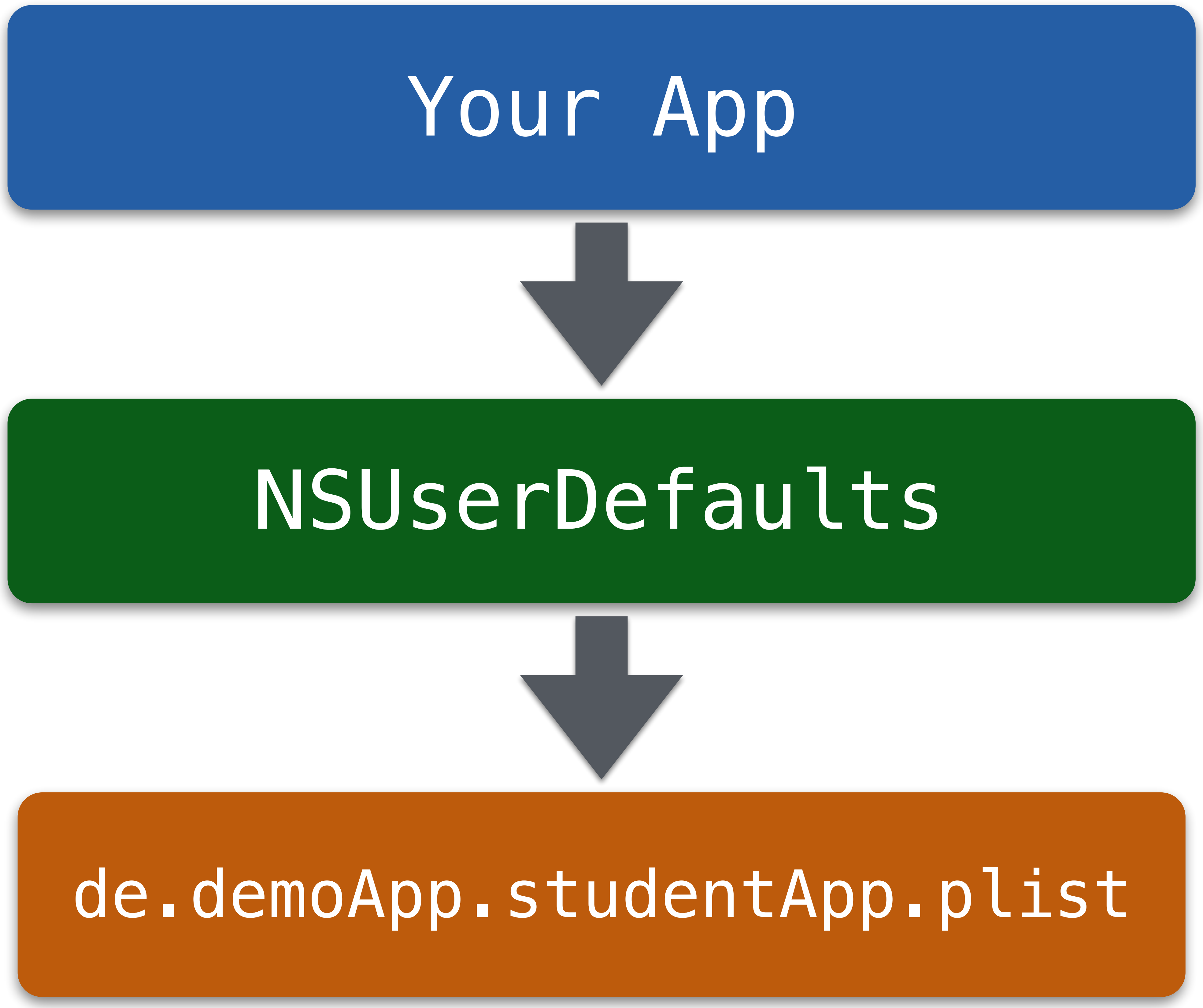
Agenda

1. **Introduction** - Organisatorisches
2. **First iOS-Project** - Hello World, **First iOS-Project** - Still Hello World (now with Code 😄)
3. **Swift**, Wait!, What about Objective-C?, Why Swift?
4. **A (not so) Quick Tour**
5. **Documentation**
6. **The basics –** iOS Architecture & more
7. **User Interfaces –** View Controller, Auto Layout & Size Classes
8. **Storyboard & Segues**
9. **Tables & NavigationController**
10. **TabBarController**
11. **Notifications**
12. **PickerViews**
13. **Touches, Gestures, 3D Touch, Peek & Pop**
14. **ScrollView & StackViews**
15. **Networking** - JSON & Dependency Managers
16. **WebKit**
17. **Maps**
18. **Storage & Data persistency –** NSUserDefaults, NSKeyedArchiver & Core Data
19. **ObjC**

Storage & Data persistency

NSUserDefaults,
NSKeyedArchiver & Core Data,

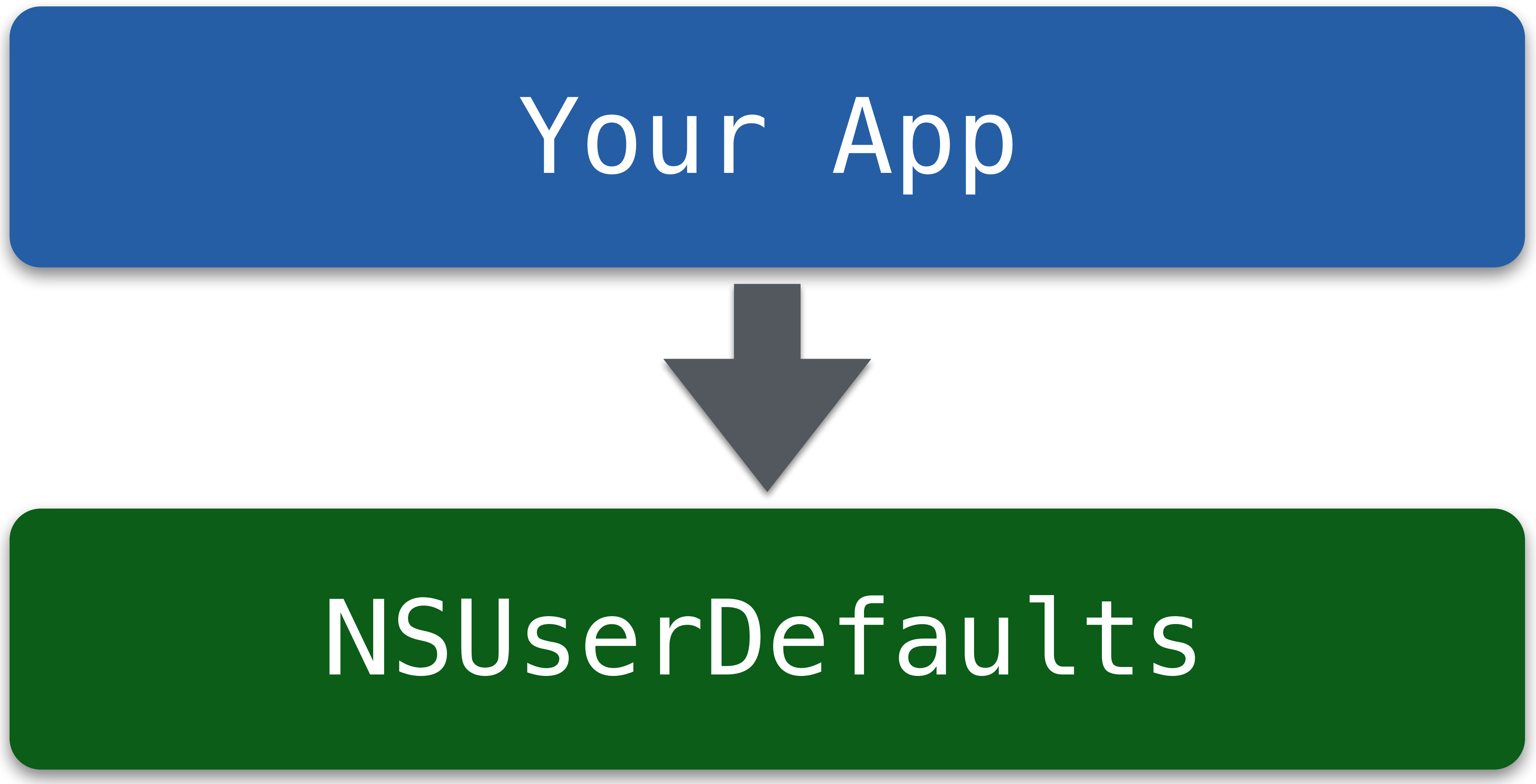
NSUserDefaults 1/4



NSUserDefaults 2/4

```
de.demoApp.studentApp.plist




name:”Hans”
```

NSUserDefaults 3/4



NSUserDefaults 4/4

```
//Save an Object

let defaults:NSUserDefaults = NSUserDefaults.standardUserDefaults()

defaults.setString(“Hans”, forKey: “name")

//synchronize the object - Important don't forget that
defaults.synchronize()

//Load an Object
//get the standardUserDefaults from the OS
let defaults: NSUserDefaults = NSUserDefaults.standardUserDefaults()

//get Name from the Defaults by Key
let name = defaults.stringForKey("name")
```

er

NSKeyedArchiver 1/5

- easy way to persistent complex data
- Use NSCoding in our Model
    - NSCoding has two methods:
        - init(coder decoder:NSCoder)
        - encodeWithCoder(coder:NSCoder)

NSKeyedArchiver 2/5

```
//init Decoder
required convenience init(coder decoder: NSCoder) {
  //super Init
  self.init()
  //take name from model and decode the variable as String
  self.name = decoder.decodeObjectForKey("name") as String
}

func encodeWithCoder(coder: NSCoder) {
  //encodeObject for the key “name”
  coder.encodeObject(self.name, forKey: "name")
}
```

ler

NSKeyedArchiver 3/5

- Specifies the directory where your data will be saved

```
//get Path for the documentDirectory for the App
var documentDirectories:AnyObject =
      NSSearchPathForDirectoriesInDomains(
                .DocumentDirectory,
                .UserDomainMask,
                true)

//init documentDirectory
var documentDirectory:String! = ""
//init Path
var path:String! = ""
```

NSKeyedArchiver 4/5

- viewDidLoad() get and create the file path

```
override func viewDidLoad() {
  super.viewDidLoad()
  //get specified path to the  documentDirectory
  documentDirectory = documentDirectories[0] as String
  //add filename and extension to the documentdirecorty path
  path = documentDirectory.stringByAppendingString("student.archive")
}
```

NSKeyedArchiver 5/5

- save Object

```
  //save Object  to path
NSKeyedArchiver.archiveRootObject(object, toFile: path)
```

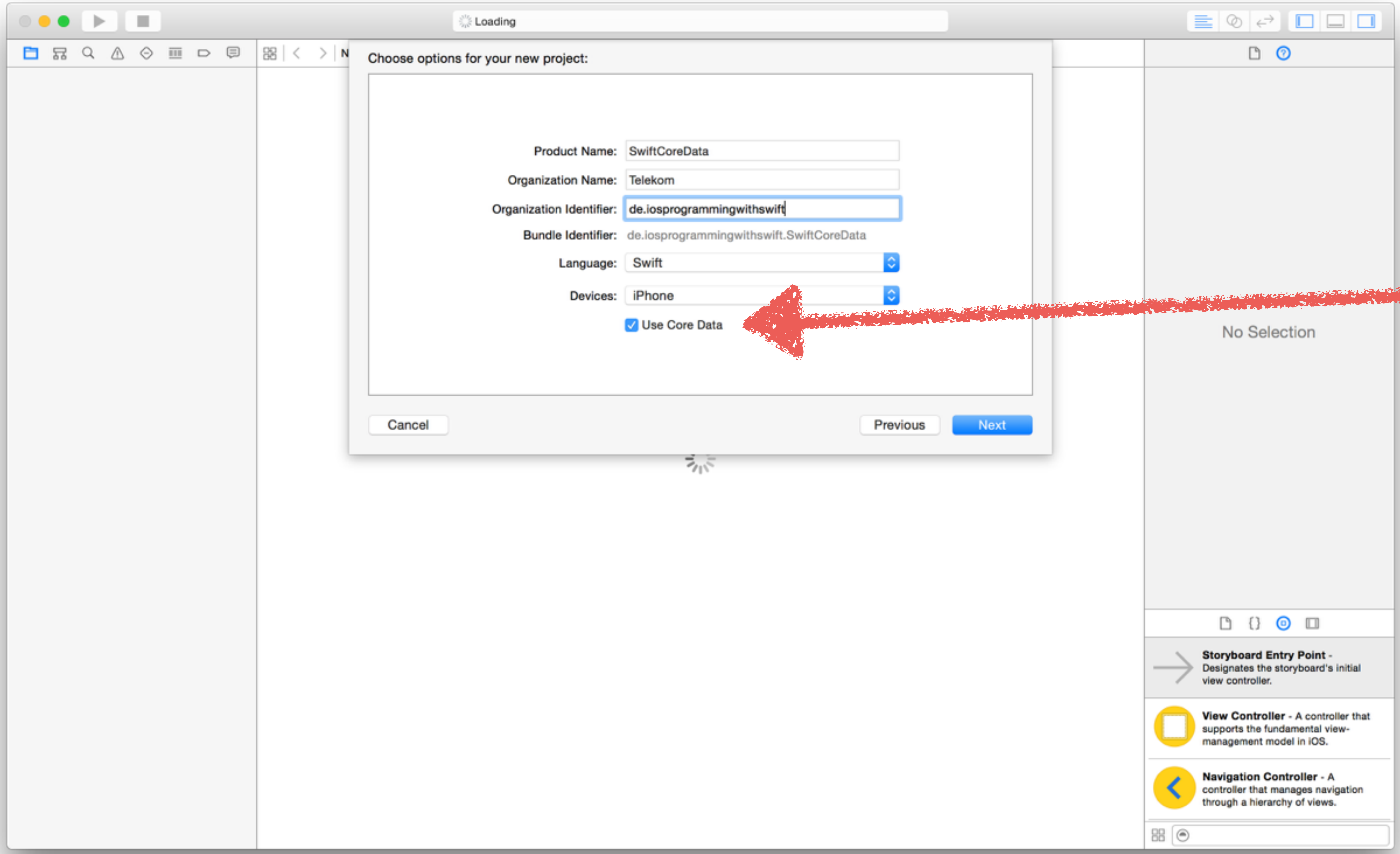
- load Object

```
  //load unspecified object from file as specified Object
NSKeyedUnarchiver.unarchiveObjectWithFile(path) as Student
```

Core Data 1/15

- modelling, querying, traversing and persistent complex data
  - supports Migrations and Undo Manager automatic
  - Serialization Format
    - SQLite, XML or NSData

Core Data 2/15

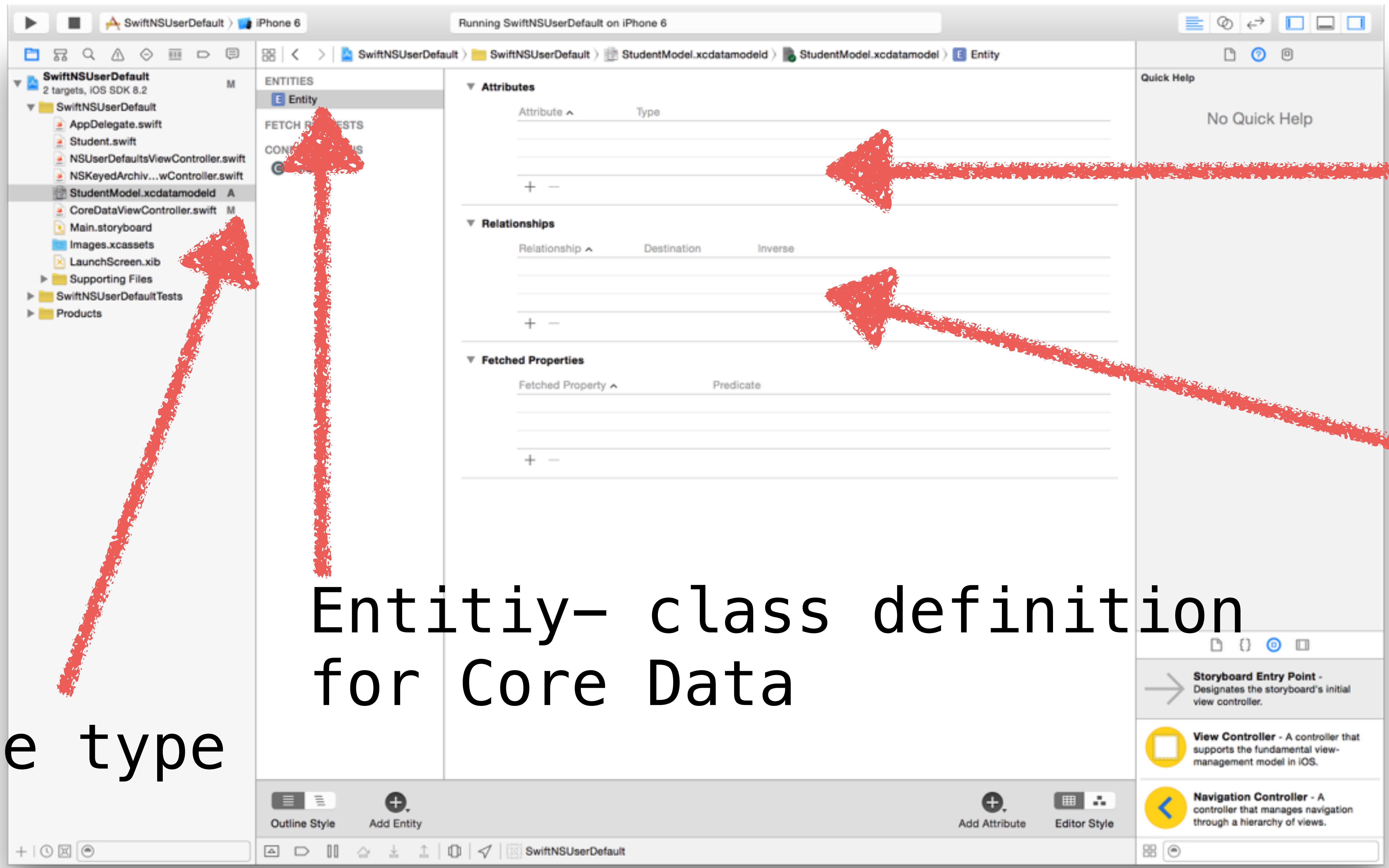

Choose options for your new project:

Product Name: SwiftCoreData
Organization Name: Telekom
Organization Identifier: de.iosprogrammingwithswift
Bundle Identifier: de.iosprogrammingwithswift.SwiftCoreData
Language: Swift
Devices: iPhone
[✓] Use Core Data

Loading
Cancel
Previous
Next
No Selection
Storyboard Entry Point - Designates the storyboard's initial view controller.
View Controller - A controller that supports the fundamental view-management model in iOS.
Navigation Controller - A controller that manages navigation through a hierarchy of views.

set “Use Core Date”
or it will much more pain in the ass

Core Data 3/15

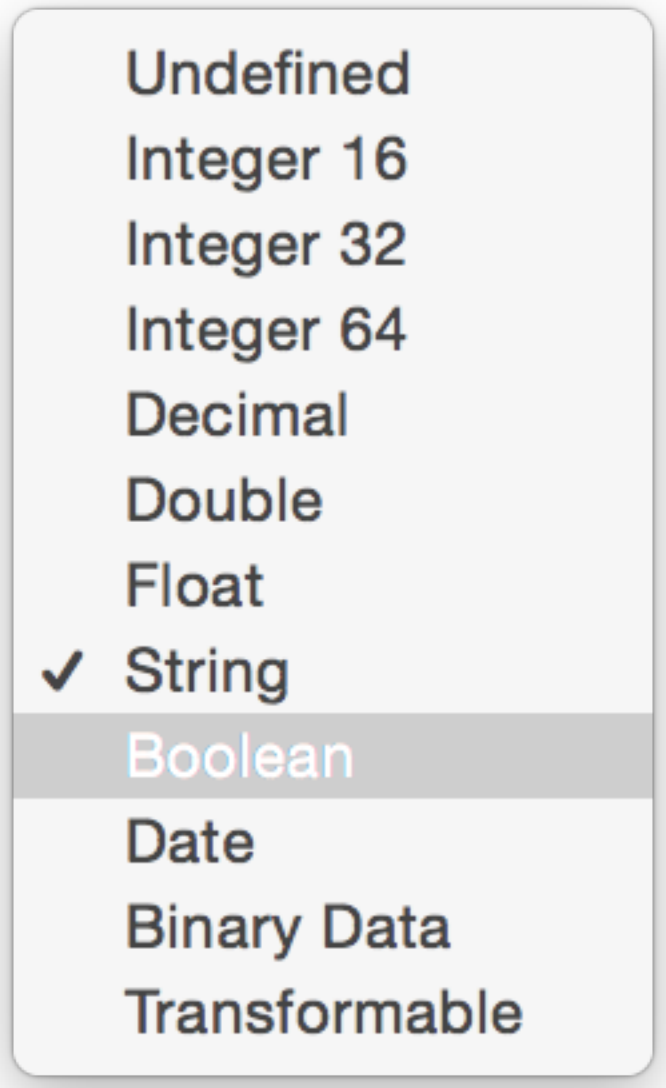

SwiftNSUserDefault
iPhone 6
Running SwiftNSUserDefault on iPhone 6
SwiftNSUserDefault
2 targets, iOS SDK 8.2
AppDelegate.swift
Student.swift
NSUserDefaultsViewController.swift
NSKeyedArchiv...wController.swift
StudentModel.xcdatamodeld
CoreDataViewController.swift
Main.storyboard
Images.xcassets
LaunchScreen.xib
Supporting Files
SwiftNSUserDefaultTests
Products
ENTITIES
Entity
FETCH REQUESTS
CONFIGURATIONS
Attributes
Attribute
Type
Relationships
Relationship
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
Quick Help
No Quick Help
Storyboard Entry Point - Designates the storyboard's initial view controller.
View Controller - A controller that supports the fundamental view-management model in iOS.
Navigation Controller - A controller that manages navigation through a hierarchy of views.
Outline Style
Add Entity
Add Attribute
Editor Style

Attributes for the class
Types: String, Int….
relationship between different entities

Entitiy– class definition for Core Data

new file type

Core Data 4/15



Attribute
s types

Core Data 5/15

- First step after creating a new Project look in the AppDelegate
  - You see Xcode generates a lot of new variables and functions to support you

```
    //The directory the application uses to store the file
    lazy var applicationDocumentsDirectory: NSURL = …
```

Check it before run/ or i
you rename your Core Data f

```
    //The managed object model for the application
  lazy var managedObjectModel: NSManagedObjectModel = let modelURL =
NSBundle.mainBundle().URLForResource("CoreDataFileName", withExtension: "momd")!
        return NSManagedObjectModel(contentsOfURL: modelURL)!
    }()
```

Core Data 6/15

```
      //The persistent store coordinator where are the files persistent saved?!
      lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator? = {var coordinator: NSPersistentStoreCoordinator? =
      NSPersistentStoreCoordinator(managedObjectModel: self.managedObjectModel)
        let url = self.applicationDocumentsDirectory.URLByAppendingPathComponent("CoreDataFileName.sqllite")
        var error: NSError? = nil
        var failureReason = "There was an error creating or loading the application's saved data."
        if coordinator!.addPersistentStoreWithType(NSSQLiteStoreType, configuration: nil, URL: url, options: nil, error: &error) == nil {
            coordinator = nil
            // Report any error we got.
            var dict = [String: AnyObject]()
            dict[NSLocalizedDescriptionKey] = "Failed to initialize the application's saved data"
            dict[NSLocalizedFailureReasonErrorKey] = failureReason
            dict[NSUnderlyingErrorKey] = error
            error = NSError(domain: "YOUR_ERROR_DOMAIN", code: 9999, userInfo: dict)
            // Replace this with code to handle the error appropriately.
            // abort() causes the application to generate a crash log and terminate. You should not use this function in a shipping
application, although it may be useful during development.
            NSLog("Unresolved error \(error), \(error!.userInfo)")
            abort()
        }

        return coordinator
    }()
```

Lazy variables will be init at there first usage

Core Data 7/15

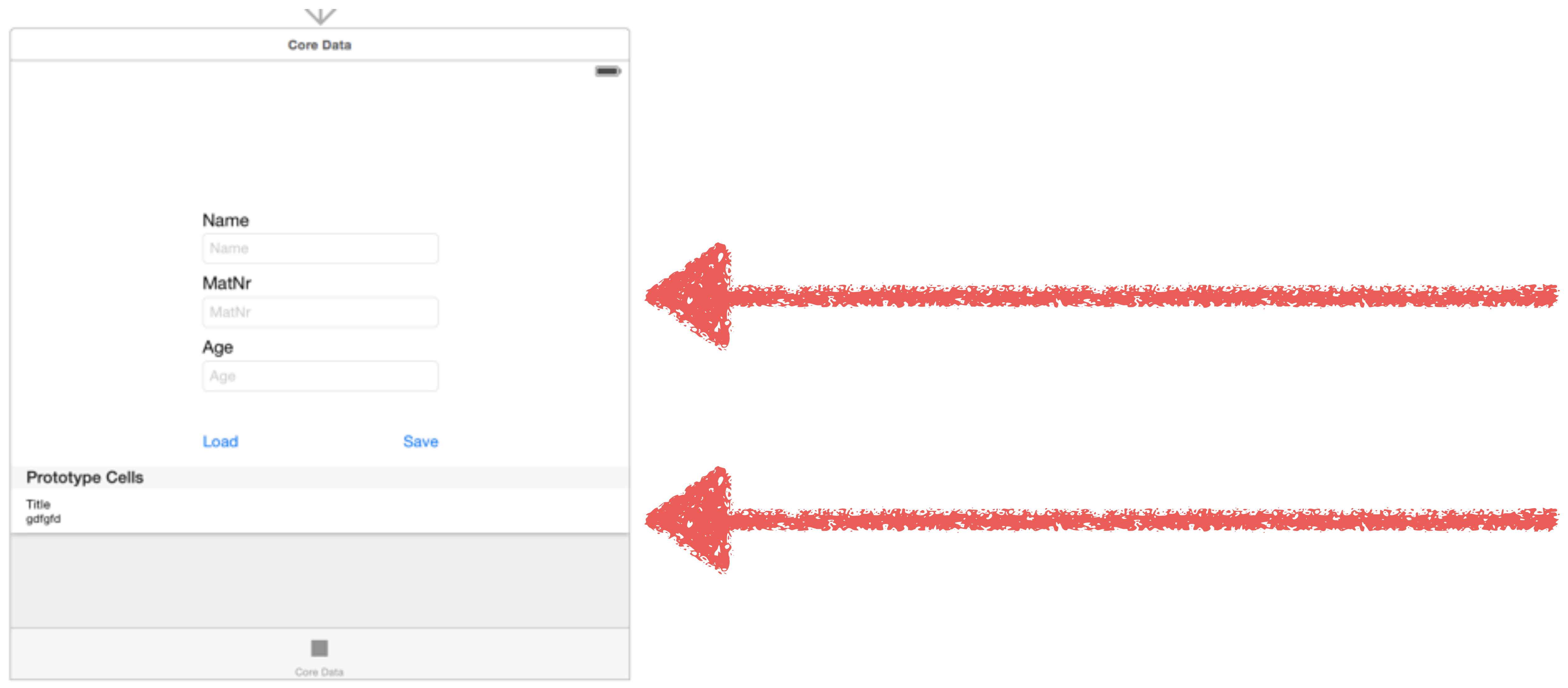
```
   // Returns the managed object context for the application - for our case we have to
  // call an mangedObjectContext on the ViewController too
lazy var managedObjectContext: NSManagedObjectContext? = {
        let coordinator = self.persistentStoreCoordinator
        if coordinator == nil {
            return nil
        }
        var managedObjectContext = NSManagedObjectContext()
        managedObjectContext.persistentStoreCoordinator = coordinator
        return managedObjectContext
        }()
```

Core Data 8/15

```
// MARK: - Core Data Saving support

    func saveContext () {
        if let moc = self.managedObjectContext {
            var error: NSError? = nil
                //error pointer
               if moc.hasChanges && !moc.save(&error)
                 NSLog("Unresolved error \(error), \(error!.userInfo)")
                abort()
            }
        }
    }
```

Core Data 9/15

- Take a look in to the ViewController



Core Data 10/15

```
 // CoreData Framework
import CoreData
// use TableViewDataSource and Delegate for data presentation
// use NSFetchedResultsControllerDelegate for database handling
class CoreDataViewController : UIViewController, UITableViewDataSource, UITableViewDelegate,
NSFetchedResultsControllerDelegate {
//Outlet for the tableView
@IBOutlet var tableView: UITableView!

//get instance from AppDelegate and use the managedObjectContext
 lazy var managedObjectContext : NSManagedObjectContext? = {
        let appDelegate = UIApplication.sharedApplication().delegate as AppDelegate
        if let managedObjectContext = appDelegate.managedObjectContext {
            return managedObjectContext
        }
        else {
            return nil
        }
        }()
```

Core Data 11/15

```
//save Button Methode
    //get Student Infos from the Textfields
    let student:Student = Student(studentName: nameTxtFld.text, studentMatNr: matNrTxtFld.text.toInt()!, studentAge: ageTxtFld.text.toInt()!)


        let context = self.fetchedResultsController.managedObjectContext
        let entity = self.fetchedResultsController.fetchRequest.entity!
        let newManagedObject = NSEntityDescription.insertNewObjectForEntityForName(entity.name!, inManagedObjectContext: context) as 
NSManagedObject

        // If appropriate, configure the new managed object.
        // Normally you should use accessor methods, but using KVC here avoids the need to add a custom class to the template.
        newManagedObject.setValue(student.name, forKey: "name")

        newManagedObject.setValue(student.matNr, forKey: "matNr")

        newManagedObject.setValue(student.age, forKey: "age")
        // Save the context.
        var error: NSError? = nil
        if !context.save(&error) {
            // Replace this implementation with code to handle the error appropriately.
            // abort() causes the application to generate a crash log and terminate. You should not use this function in a shipping 
application, although it may be useful during development.
            //println("Unresolved error \(error), \(error.userInfo)")
            abort()
        }
```

Core Data 12/15

```swift
var fetchedResultsController: NSFetchedResultsController {
        if _fetchedResultsController != nil {
            return _fetchedResultsController!
        }

        let fetchRequest = NSFetchRequest()
        // Edit the entity name as appropriate.
        let entity = NSEntityDescription.entityForName("CoreStudent", inManagedObjectContext: self.managedObjectContext!)
        fetchRequest.entity = entity

        // Set the batch size to a suitable number.
        fetchRequest.fetchBatchSize = 20

        // Edit the sort key as appropriate.
        let sortDescriptor = NSSortDescriptor(key: "name", ascending: false)
        let sortDescriptors = [sortDescriptor]

        fetchRequest.sortDescriptors = [sortDescriptor]
```

Core Data 13/15

```
        // Edit the section name key path and cache name if appropriate.
        // nil for section name key path means "no sections".
        let aFetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest, managedObjectContext: 
self.managedObjectContext!, sectionNameKeyPath: nil, cacheName: "Master")
        aFetchedResultsController.delegate = self
        _fetchedResultsController = aFetchedResultsController

        var error: NSError? = nil
        if !_fetchedResultsController!.performFetch(&error) {
            // Replace this implementation with code to handle the error appropriately.
            // abort() causes the application to generate a crash log and terminate. You should not use this function in a shipping 
application, although it may be useful during development.
            //println("Unresolved error \(error), \(error.userInfo)")
            abort()
        }

        return _fetchedResultsController!
    }


    var _fetchedResultsController: NSFetchedResultsController? = nil
```

Core Data 14/15

```
 func controllerWillChangeContent(controller: NSFetchedResultsController) {
        self.tableView.beginUpdates()
    }

    func controller(controller: NSFetchedResultsController, didChangeSection sectionInfo: NSFetchedResultsSectionInfo, atIndex sectionIndex: Int, forChangeType type:
NSFetchedResultsChangeType) {
        switch type {
        case .Insert:
            self.tableView.insertSections(NSIndexSet(index: sectionIndex), withRowAnimation: .Fade)
        case .Delete:
            self.tableView.deleteSections(NSIndexSet(index: sectionIndex), withRowAnimation: .Fade)
        default:
            return
        }
    }
 func controller(controller: NSFetchedResultsController, didChangeObject anObject: AnyObject, atIndexPath indexPath: NSIndexPath?, forChangeType type: NSFetchedResultsChangeType,
newIndexPath: NSIndexPath?) {
        switch type {
        case .Insert:
            tableView.insertRowsAtIndexPaths([newIndexPath!], withRowAnimation: .Fade)
        case .Delete:
            tableView.deleteRowsAtIndexPaths([indexPath!], withRowAnimation: .Fade)
        case .Move:
            tableView.deleteRowsAtIndexPaths([indexPath!], withRowAnimation: .Fade)
            tableView.insertRowsAtIndexPaths([newIndexPath!], withRowAnimation: .Fade)
        default:
            return
        }
    }

    func controllerDidChangeContent(controller: NSFetchedResultsController) {
        self.tableView.endUpdates()
    }
```

NSFetchedResultsController shows us the status of the database operations

Core Data15/5

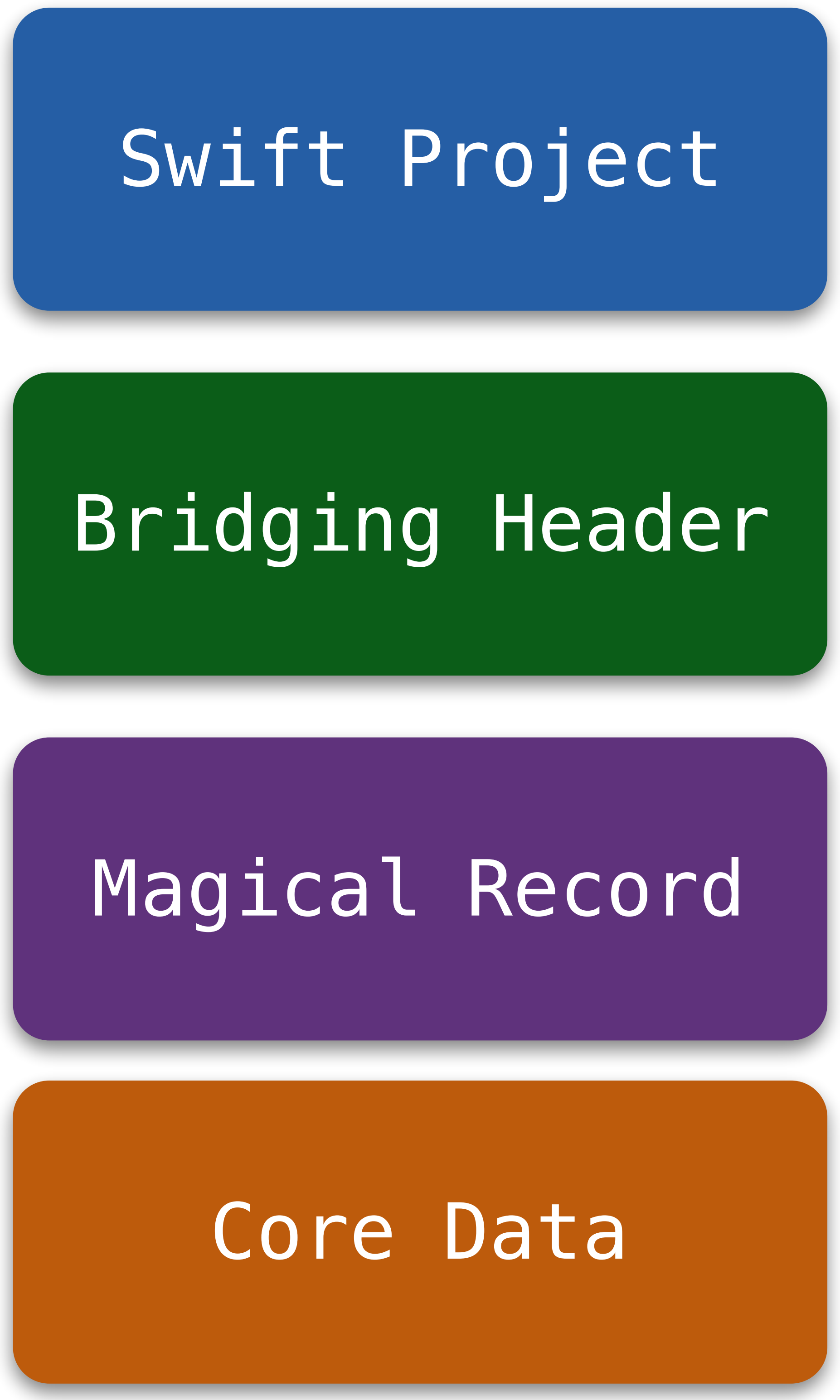
```
 func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {

        // get the sectionInfo from the fetchedresultController
            let sectionInfo = self.fetchedResultsController.sections![section] as
NSFetchedResultsSectionInfo
            return sectionInfo.numberOfObjects
    }

    func tableView(tableView: UITableView,cellForRowAtIndexPath  indexPath: NSIndexPath) ->
UITableViewCell {

        let cell = tableView.dequeueReusableCellWithIdentifier("StudentCell", forIndexPath: indexPath)
as UITableViewCell

        //configureCell
        let student = self.fetchedResultsController.objectAtIndexPath(indexPath) as NSManagedObject
        cell.textLabel!.text = student.valueForKey("name")!.description
        cell.detailTextLabel!.text = student.valueForKey("matNr")!.description
        return cell
    }
```

Magical Record 1/4

- Open Source and available over CocoaPods

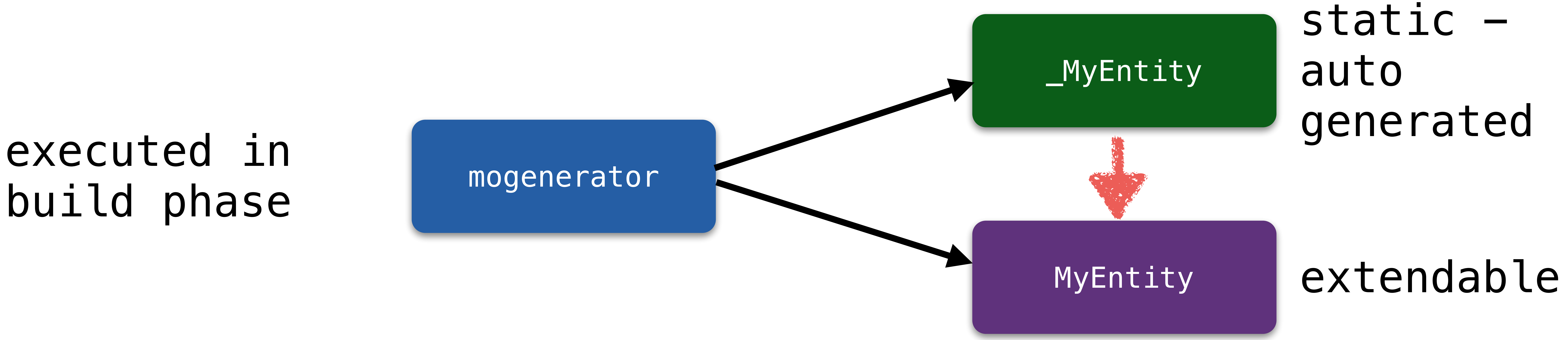
e

Active Record is the M in MVC – the model – which is the layer of the system responsible for representing business data and logic. Active Record facilitates the creation and use of business objects whose data requires persistent storage to a database.
Quelle:
guides.rubyonrails.org/active_record_basics.html

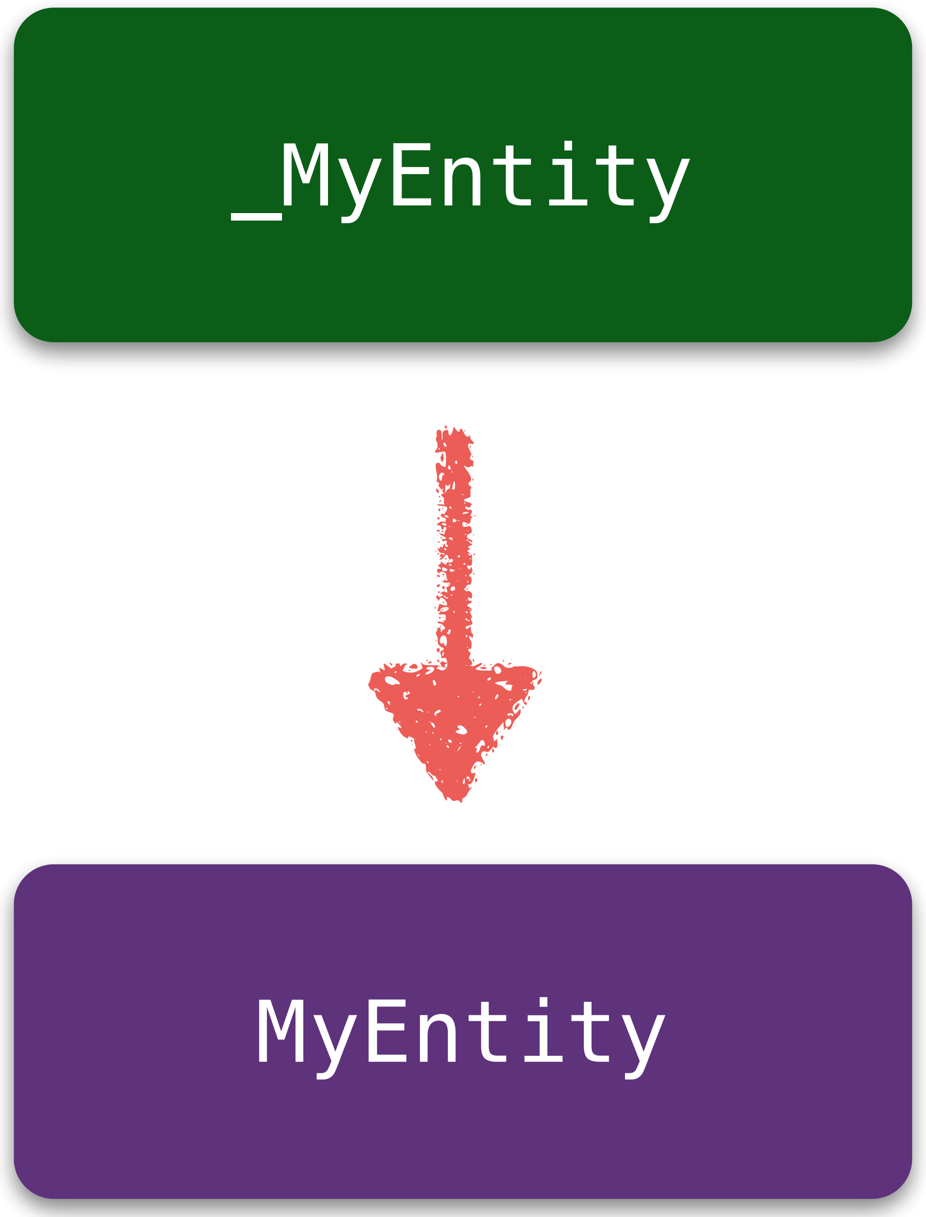


Magical Records 2/4

- mogenerator - tool for automtacliy generating classes from entities
  - Generates two classes per entity



Magical Records 3/4

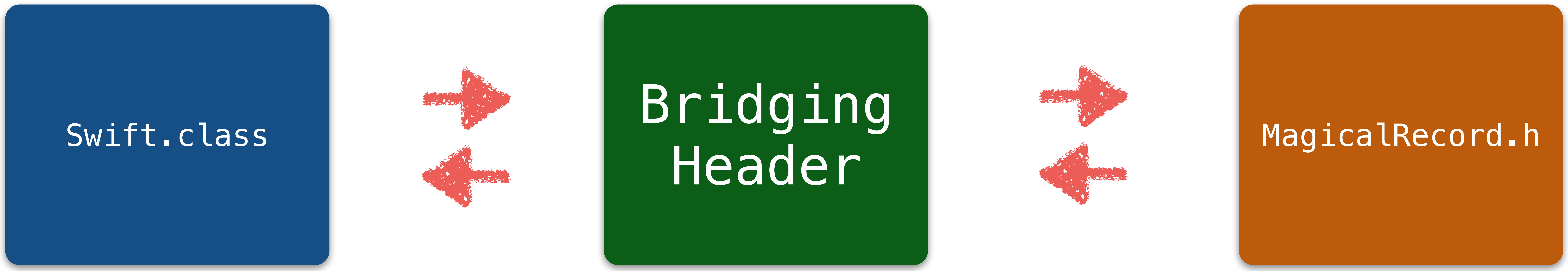


automatically updated
in build phase

Subclass of _MyEntity
Custom Logic

Magical Records 4/4

- Bridging Headers make it possible to using Objective-C Code in Swift Project.

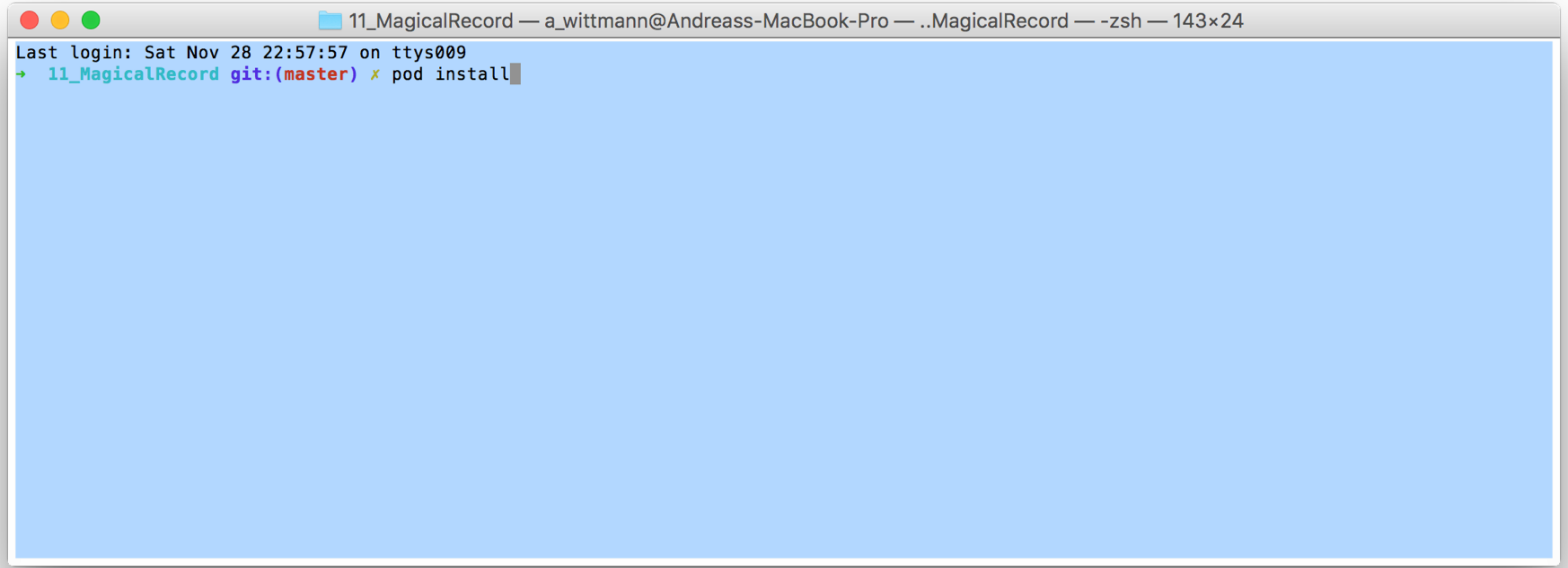


Magical Record Demo 1/0

- open MagicalRecord Project
- open terminal
- navigate to the project

Magical Record Demo 2/0

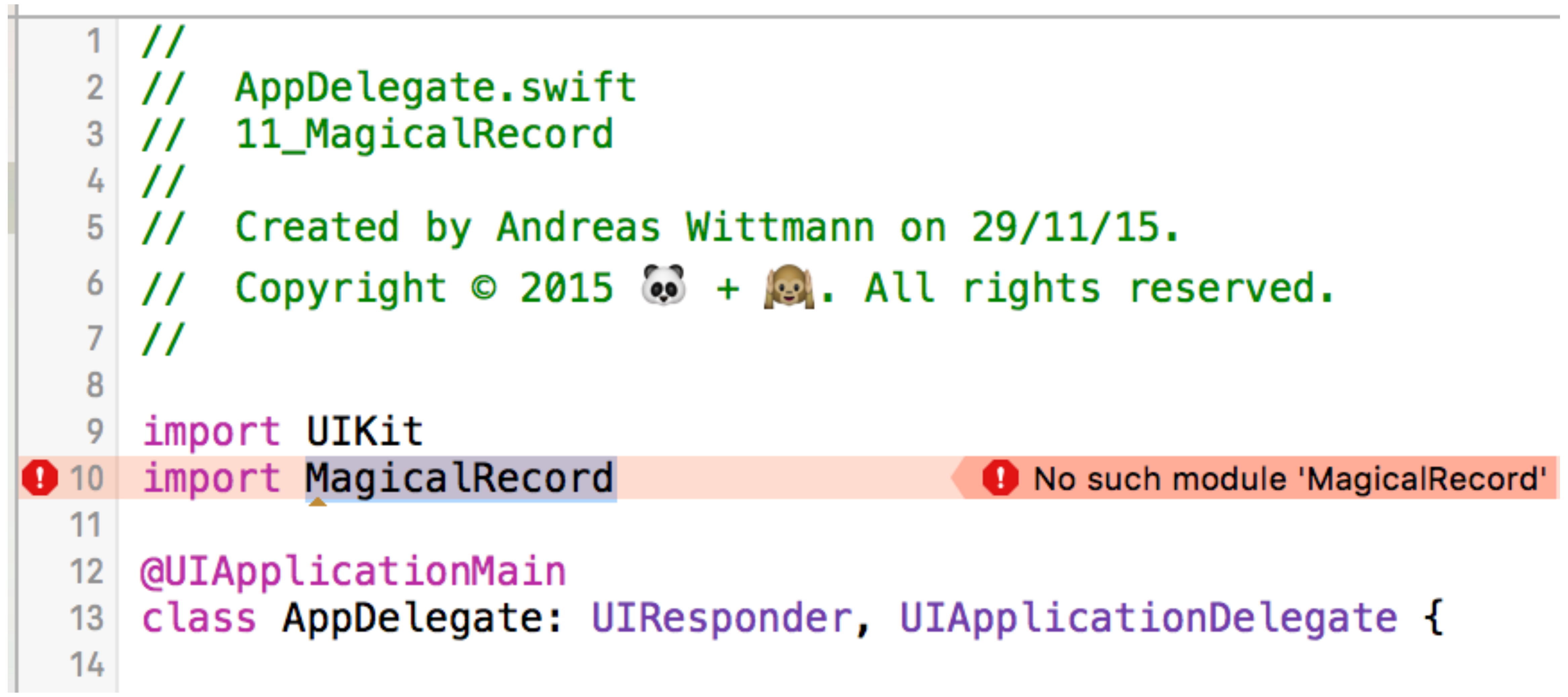
- pod install


11_MagicalRecord — a_wittmann@Andreass-MacBook-Pro — ..MagicalRecord — -zsh — 143×24

```
Last login: Sat Nov 28 22:57:57 on ttys009
➜  11_MagicalRecord git:(master) ✗ pod install
```

Magical Record Demo 3/0

- open 11_MagicalRecord.xcworkspace
- AppDelegate.swift:
  import MagicalRecord



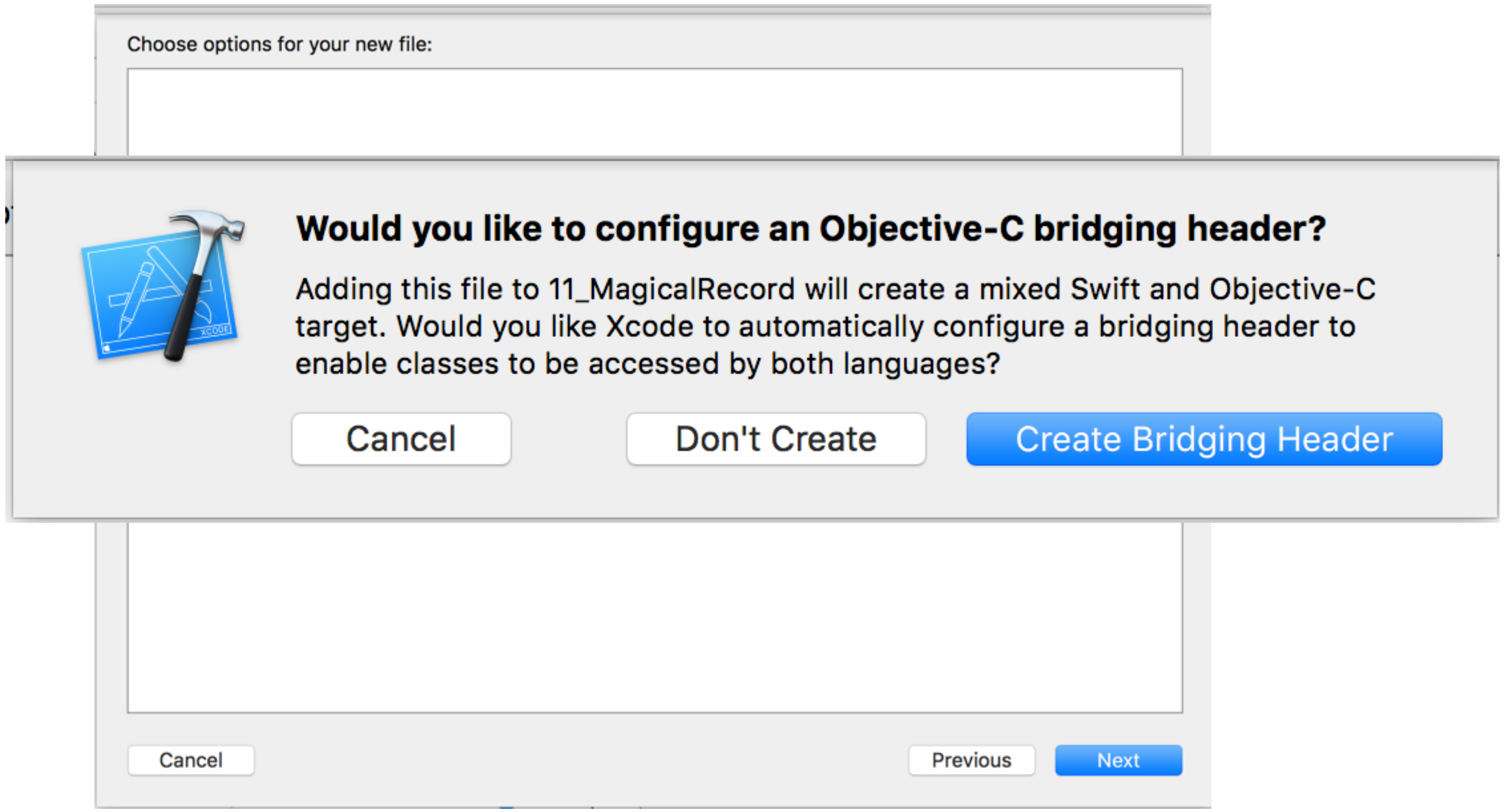
```
//
//  AppDelegate.swift
//  11_MagicalRecord
//
//  Created by Andreas Wittmann on 29/11/15.
//  Copyright © 2015 🐼 + 🙈. All rights reserved.
//

import UIKit
import MagicalRecord                          ❗ No such module 'MagicalRecord'

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

```

Swift Project	✅
Bridging Header	❌
Magical Record	✅
Core Data	

Magical Record Demo 4/0

- add new File (⌘ + N)

Bridging Header



Choose options for your new file:

Would you like to configure an Objective-C bridging header?

Adding this file to 11_MagicalRecord will create a mixed Swift and Objective-C target. Would you like Xcode to automatically configure a bridging header to enable classes to be accessed by both languages?

Magical Record Demo 5/0

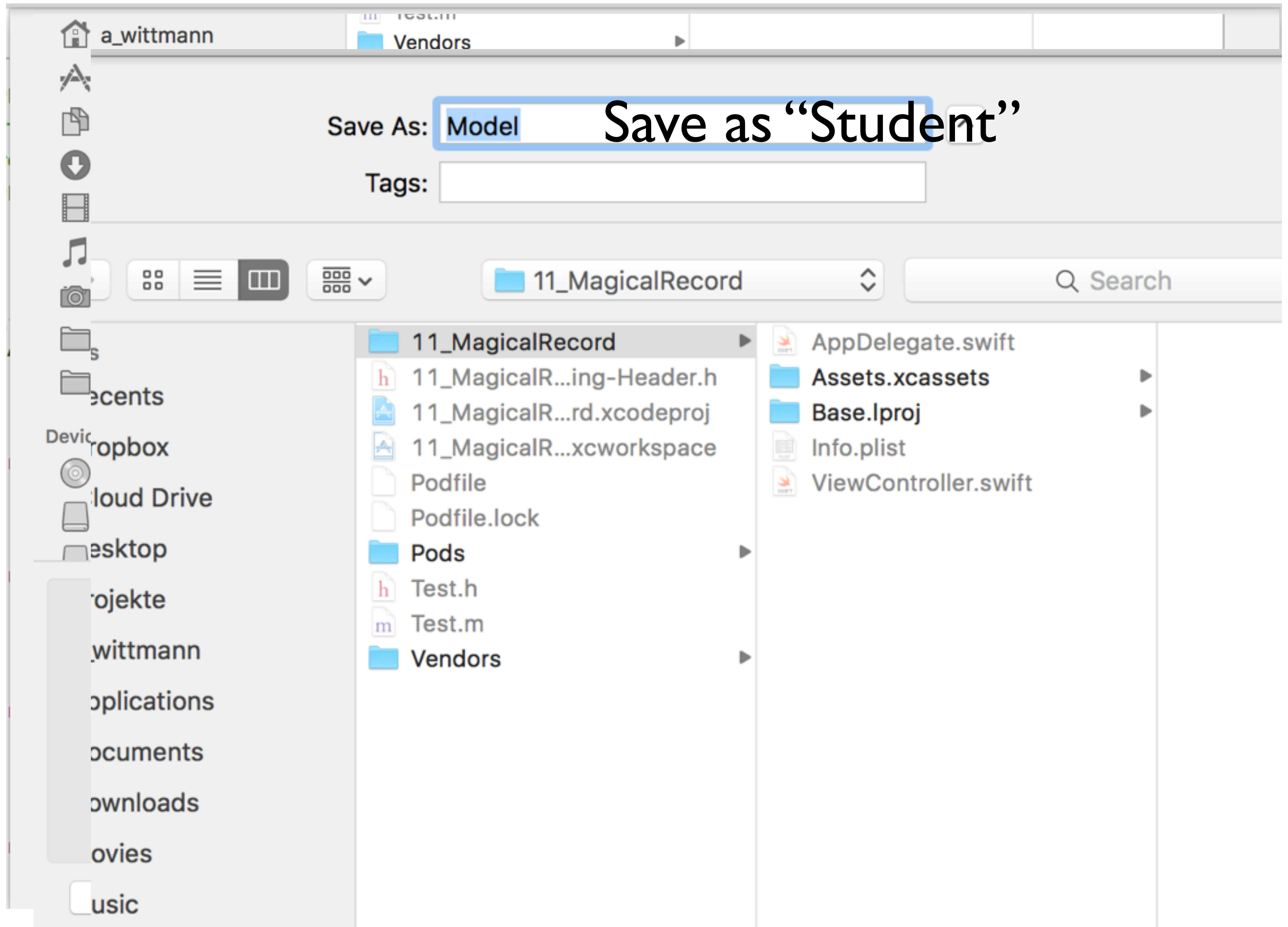
- open 11_MagicalRecord-Bridging-Header.h

```
#import <MagicalRecord/MagicalRecord.h>
```

Swift Project	✅
Bridging Header	❌
Magical Record	✅
Core Data	

Magical Record Demo 6/0

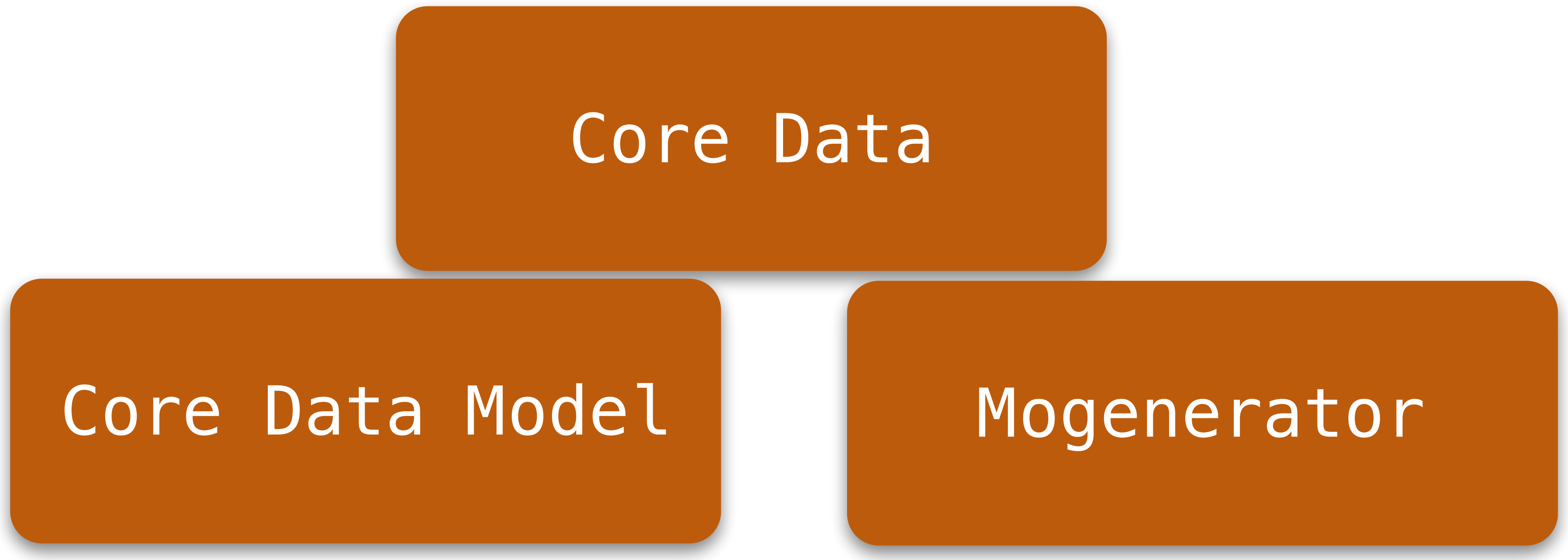
- add new File (⌘ + N)


Save As: Model

Save as "Student"

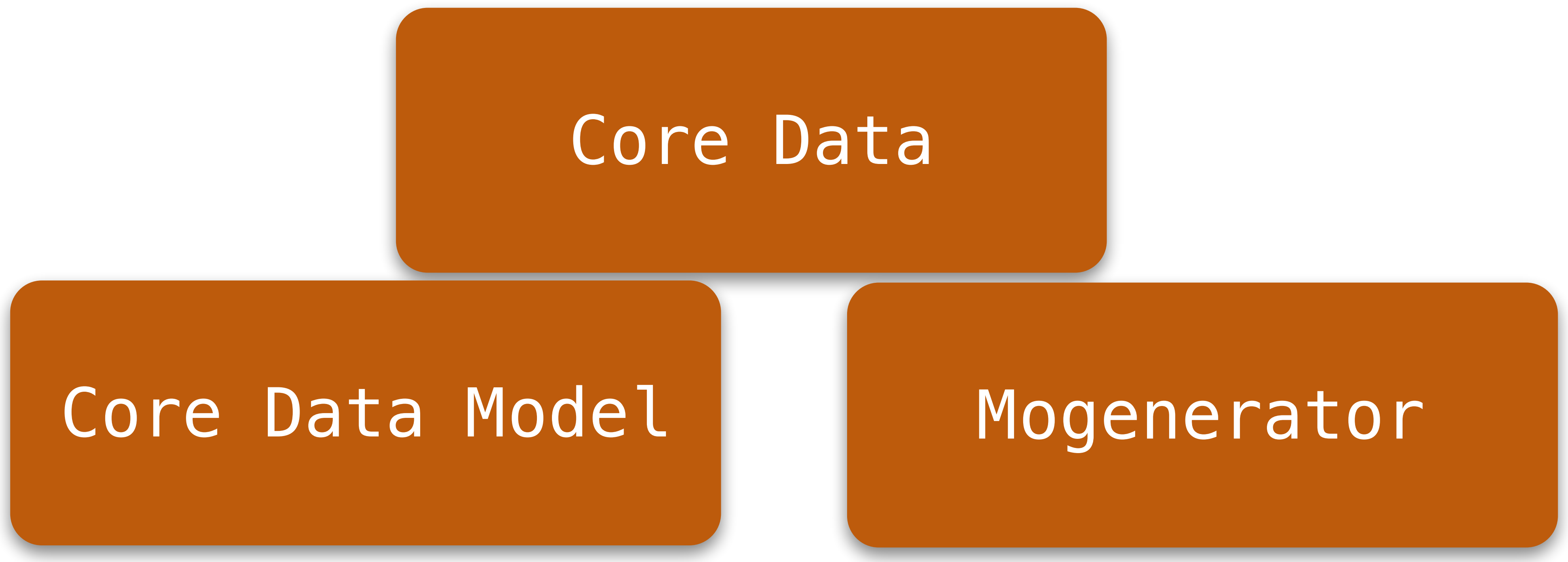
Tags:

11_MagicalRecord

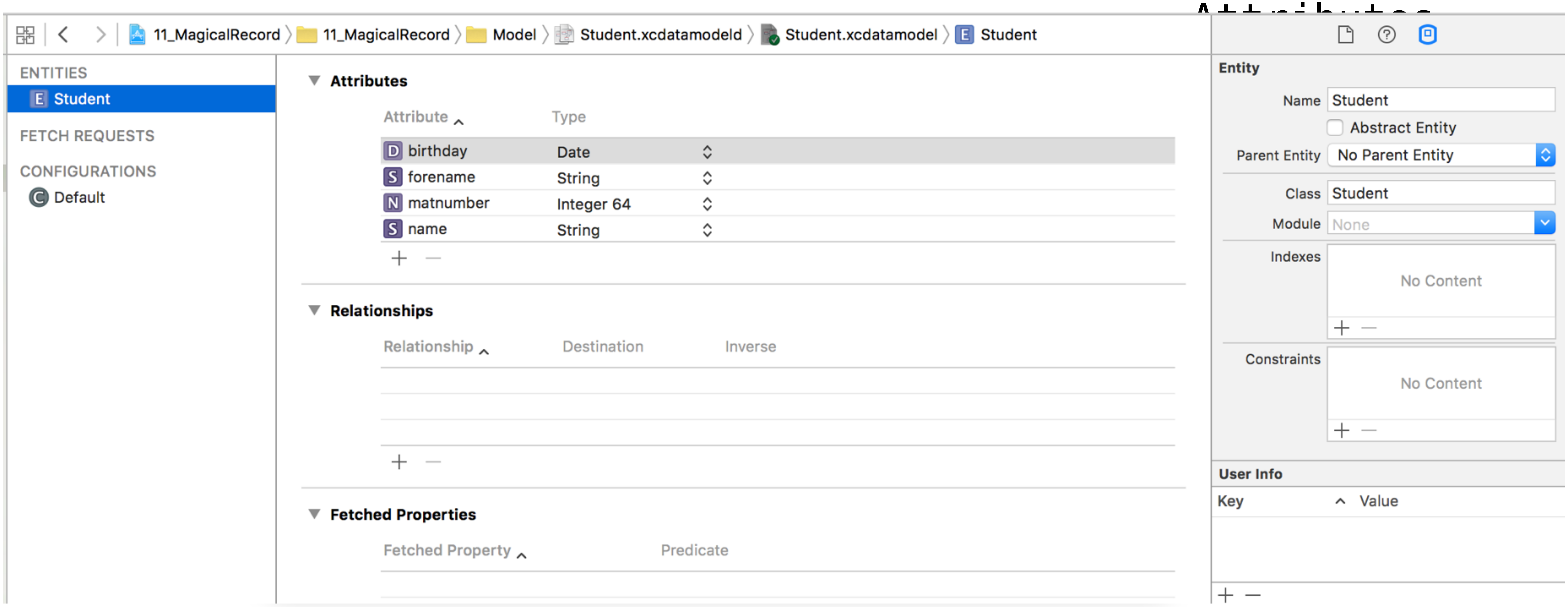


Magical Record Demo 7/0

- New Group: Student.xcdatamodeld
- Add Entity: Student.xcdatamodeld



Magical Record Demo 8/0



11_MagicalRecord 〉 11_MagicalRecord 〉 Model 〉 Student.xcdatamodeld 〉 Student.xcdatamodel 〉 Student

ENTITIES
- Student

FETCH REQUESTS

CONFIGURATIONS
- Default

Attributes

Attribute	Type
D birthday	Date
S forename	String
N matnumber	Integer 64
S name	String

Relationships

Relationship	Destination	Inverse

Fetched Properties

Fetched Property	Predicate

Entity

Name	Student
	Abstract Entity
Parent Entity	No Parent Entity
Class	Student
Module	None
Indexes	No Content
Constraints	No Content

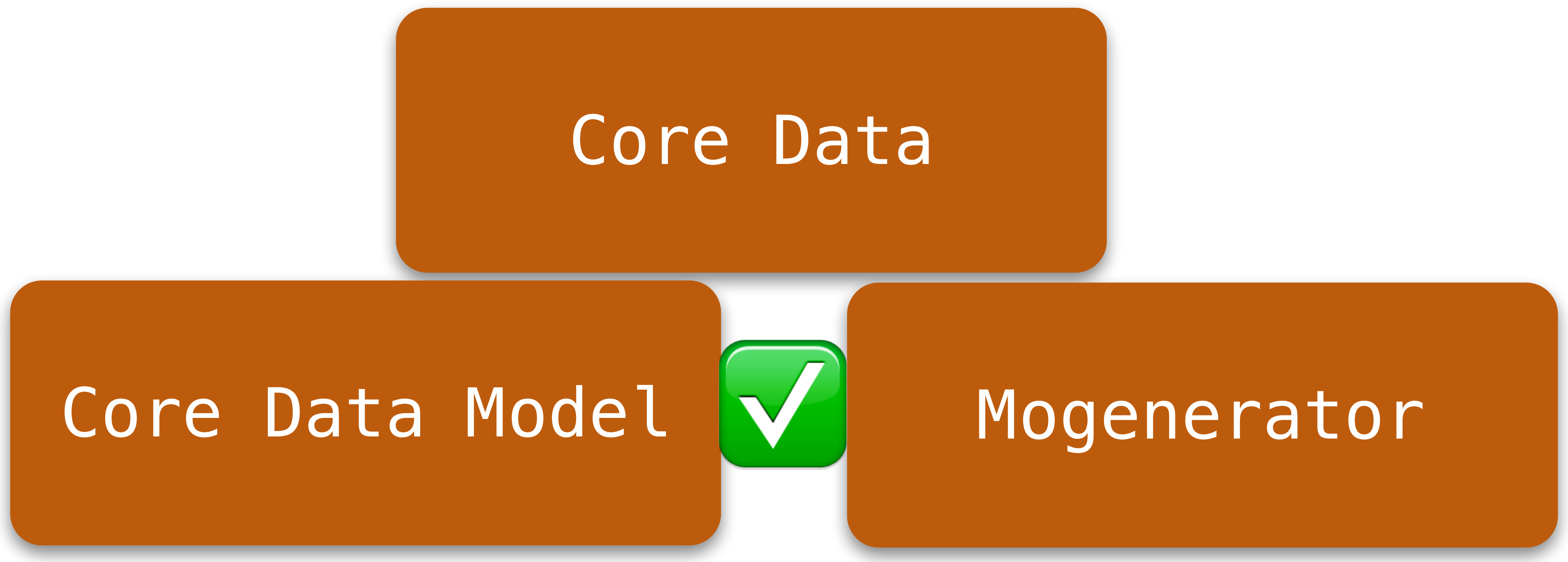
User Info

Key	Value

Magical Record Demo 9/0

- Files Add to:
  Vendors

- Vendors:
  removing Target
  Membership



Magical Record Demo 10/0

- open /vendors/Mogenerator/mo.command
- change:

```
MagicalRecordTest
```

```
11_MagicalRecord
```

Mogenerator

Magical Record Demo 11/0

- change:

Mogenerator

Model.xcd…

Student.xcd…

Magical Record Demo 12/0

- open Project Settings
- Click on Build Phases

Mogenerator

11_MagicalRecord
Vendors
Mogenerator
mo.command
mogenerator
templates
human.swift.motemplate
machine.swift.motemplate
11_MagicalRecord
AppDelegate.swift
Model
Student.xcdatamodeld
ViewController.swift
General
Capabilities
Resource Tags
Info
Build Settings
Build Phases
Build Rules
PROJECT
11_MagicalRecord
TARGETS
11_MagicalRecord
11_MagicalRecord_Deb...
Filter

Target Dependencies (0 items)
Check Pods Manifest.lock
Compile Sources (3 items)
Link Binary With Libraries (1 item)
Copy Bundle Resources (3 items)

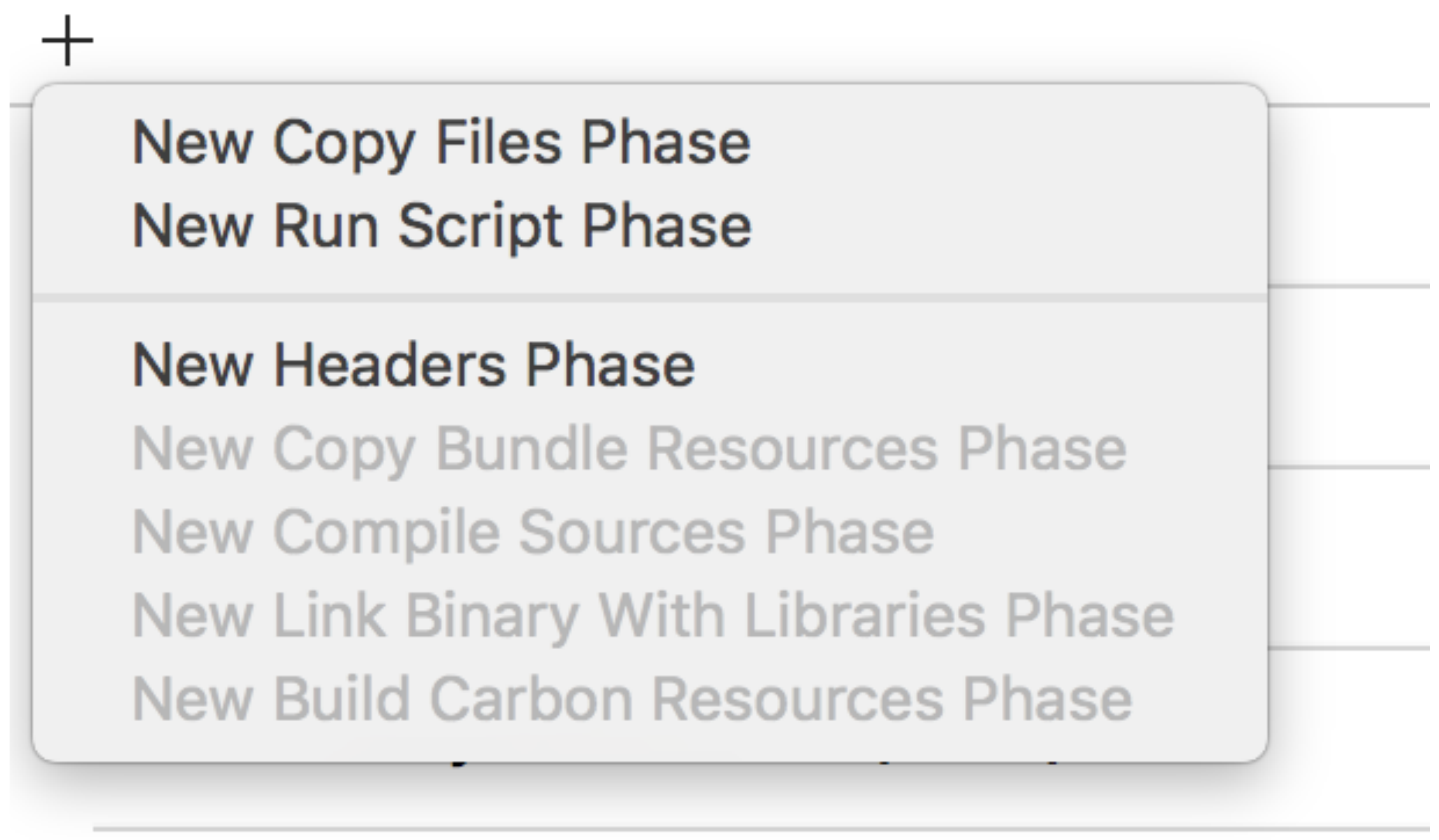
Identity and Type
Name: 11_MagicalRecord
Location: Relative to Group
11_MagicalRecord.xcodeproj
Full Path: /Users/a_wittmann/Documents/Projekte/vorlesung2/11_MagicalRecord/11_MagicalRecord.xcodeproj

Project Document
Project Format: Xcode 3.2-compatible
Organization: 🐼 + 🙈

Magical Record Demo 13/0

- Click on the Puls to add a new Run script

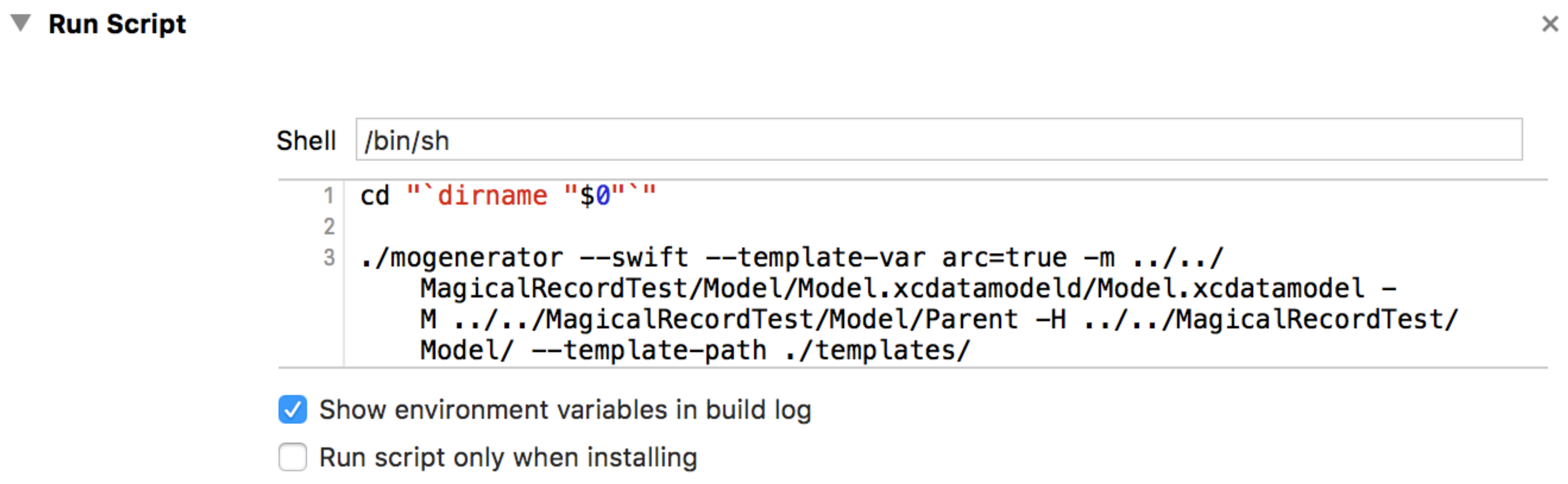
Mogenerator


+
New Copy Files Phase
New Run Script Phase
New Headers Phase
New Copy Bundle Resources Phase
New Compile Sources Phase
New Link Binary With Libraries Phase
New Build Carbon Resources Phase

Magical Record Demo 14/0

- Copy Script from /vendors/Mogenerator/mo.command

Mogenerator



Run Script

Shell /bin/sh

```
cd "`dirname "$0"`"

./mogenerator --swift --template-var arc=true -m ../../
    MagicalRecordTest/Model/Model.xcdatamodeld/Model.xcdatamodel -
    M ../../MagicalRecordTest/Model/Parent -H ../../MagicalRecordTest/
    Model/ --template-path ./templates/
```

[✓] Show environment variables in build log

[ ] Run script only when installing

Magical Record Demo 15/0

- change:

Model.xcd…

Student.xcd…

- Run mogenerator

Mogenerator

Magical Record Demo 16/0

- Files Add to:
  Parent/_Student.swift
  Student.swift

- import Foundation:
  Student.swift

Mogenerator ✅

Magical Record Demo 17/0

- now some magic

Swift Project ✅

Bridging Header ✅

Magical Record ✅

Core Data ✅

Magical Record Demo 18/0

- open file Student.swift

```
func stundenSaysHello(){
        print("Hello")
    }
```

Magical Record Demo 19/0

- open file AppDelegate.swift

```
 func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject: AnyObject]?) -> Bool {
        let defaults = NSUserDefaults.standardUserDefaults()
        if !defaults.boolForKey("FirstLaunch") {

MagicalRecord.setupCoreDataStackWithAutoMigratingSqliteStoreNamed("Student")
            setDefaultData()
            defaults.setBool(true, forKey: "FirstLaunch")
        }
        return true
    }
```

Magical Record Demo 20/0

- for synchronic saving we are using

```
NSManagedObjectContext.MR_defaultContext().MR_saveToPersistentStoreAndWait()
```

- for async saving we would using

```
NSManagedObjectContext.MR_defaultContext().MR_saveInBackgroundCompletion({
 completion handler
})
```

Magical Record Demo 21/0

- open Storyboard
  adding TableView into the ViewController
  setting Delegate and Datasource
  generating IBOutlet